Preprint typeset in JHEP style - HYPER VERSION RUNHETC-06-07



# Vscape V1.1.0
# An Interactive Tool for Metastable Vacua

**Korneel van den Broek**

*Department of Physics and NHETC*
*Rutgers University*
*Piscataway, NJ 08854*
*E-mail:* korneel@physics.rutgers.edu

Abstract: Vscape is an interactive tool for studying the one-loop effective potential of an ungauged supersymmetric model of chiral multiplets. The program allows the user to define a supersymmetric model by specifying the superpotential. The F-terms and the scalar and fermionic mass matrices are calculated symbolically. The program then allows you to search numerically for (meta)stable minima of the one-loop effective potential. Additional commands enable you to further study specific minima, by e.g. computing the mass spectrum for those vacua. Vscape combines the flexibility of symbolic software, with the speed of a numerical package.

## Contents

---

## 1. Introduction

The requirement that supersymmetry is broken at the global minima in field theories with dynamical supersymmetry breaking[1] puts tight constraints on model building. Recently, dynamical supersymmetry breaking in metastable vacua has attracted attention as a promising phenomenological possibility. This greatly simplifies the model building problems. The metastable states are viable candidates to describe our world, when the tunneling probability to the supersymmetric ground state is highly suppressed. As argued in [2], metastable vacua might be quite generic in supersymmetric theories[3][4]. With some mild assumptions, low energy supersymmetry breaking even requires our world as we know it to be a long lived metastable state[5]. From the string theory point of view, non-supersymmetric flux compactifications might also be rather generic in the landscape of vacua[6].

An important role in studying the moduli space of supersymmetric theories is played by the one-loop effective potential of the field theory. The classical (pseudo)moduli space can contain flat directions. The one-loop corrections often lift those pseudomoduli directions of the classical theory, thus creating isolated (meta)stable vacuum. Generically, several fields will have vacuum expectation values (vevs) at those local minima. The important phenomenological objective is then to create models that communicate this dynamical breaking of supersymmetry to the supersymmetric standard model such that one gets attractive superpartner spectra[7].

The computation of the Coleman-Weinberg potential[8] in realistic models often becomes rather involved as the mass matrices become large and one can expect several fields to gain vevs. Even for symbolic software package, multiple diagonalization of large mass matrices is rather time consuming, while numerical packages usually do not provide the flexibility of working with symbolic formulas. `Vscape` aims to bridge this gap between symbolic and numerical packages. The flexibility of the `Vscape` users interface gives the user the freedom to define the physics of the model symbolically, while the computationally intensive evaluations of the Coleman-Weinberg potential rely on fast numerical C++ routines.

In addition `Vscape` provides the necessary tools to analyse a metastable vacuum once one is found. Commands are provided to compute the mass spectrum at the local minimum. Those results can then be used as input data for a spectrum generator such as `SOFTSUSY`[9].

Vscape has a command implemented with allows for the creation of files formatted according the Model Input File standard described in the Les Houches Accord 1.0. to supply the high energy input to the spectrum generator.

The code for `Vscape` was written in object-oriented C++ since this language compiles to speed efficient programs, allowing for fast diagonalization of large matrices for the Coleman-Weinberg potential. Using C++ also has the advantage that an extensive collection very powerful, mathematical libraries are freely available. `Vscape` makes use of the recent libraries GSL [10], GiNaC and CLN [11]. `Vscape` also has Tab-completion build in, if one has the readline library installed. The program can freely be downloaded and installed from either:

`http://www.physics.rutgers.edu/~korneel/vscape/vscape.html`

or

`http://projects.hepforge.org/vscape/`

Detailed installation instruction (Linux and Windows) can also be found there. Updates and fixes of possible errors will also appear on those websites.

In section 2, we briefly introduce the physics behind the commands of `Vscape`. We then give a short account on how the program is structured in section 3. Section 4 lists several possibilities for further extensions of the program. Appendix A and B contain the detailed syntax and functionality of all the commands understood by `Vscape`. Section B.8 discusses the precision of the numerical computations, while section B.9 lists the commands to change the control parameters of the program to influence the algorithms underlying the program.

## 2. Physics Overview

### 2.1 The supersymmetric model of chiral fields

The current version of `Vscape` allows one to study various $\mathcal{N} = 1$ supersymmetric models with chiral fields[12]. The user defines the model by specifying the parameters and chiral fields $\phi_i$ of the model. In addition, one has to specify the subset of fields which are a priori allowed to obtain a vev for their scalar component. Let us denote this subset of background fields as $\varphi_i$, while we will use the symbol $\tilde{\varphi}_n$ for the fields that are not allowed to get a scalar vev. The model is then defined through the superpotential $W$ and the D-term potential $V_D$. The D-term potential is only taken into account in the tree level potential[1]. The current version of the program does not include terms from vector multiplets in the Coleman-Weinberg potential computation. The program assumes a canonical Kähler potential,

$$K = \mathrm{Tr}\, \phi^\dagger \phi \,. \tag{2.1}$$

[1]We include the D-terms in the tree level potential to accomodate models like [13] where the hidden sector itself does not have a D-term potential but where the Supersymmetric Standard Model (SSM) sector does contribute D-terms to the tree level potential. This also allows us to described the metastable vacua of the models studied in [2] where the theory is gauged, but the effect of the gauge fields drops out of the one-loop correction.

When a new model is loaded, the program computes and stores the following information symbolically:

$$V_0 = W_i^*|_{\tilde{\varphi}_n=0}\, W^i\big|_{\tilde{\varphi}_n=0} + V_D|_{\tilde{\varphi}_n=0} \tag{2.2}$$

$$Fterms_i = W_i|_{\tilde{\varphi}_n=0} \tag{2.3}$$

$$mF_{ij} = W_{ij}|_{\tilde{\varphi}_n=0} \tag{2.4}$$

$$mB_{ij} = W^*_{ijk}W^k\Big|_{\tilde{\varphi}_n=0} \tag{2.5}$$

where an index $i$ to the superpotential $W$ stands for a derivative with respect to field $\phi_i$. The $|_{\tilde{\varphi}_n=0}$ indicates that the fields that are not allowed to get a vev by the user, are set to zero in the expression. The commands detailed in section B.2 give you access to this information.

### 2.2 Coleman-Weinberg computation

To compute the Coleman-Weinberg potential, the numerical values for the vevs and parameters are substitued into the matrices $mF$ and $mB$. The program then constructs the numerical mass squared matrices

$$M_{1/2}^\dagger M_{1/2} = W^*_{jk}W^{ik} \tag{2.6}$$

$$M_0^2 = \begin{pmatrix} W^*_{jk}W^{ik} & W^*_{ijk}W^k \\ W^{ijk}W^*_k & W^*_{ik}W^{jk} \end{pmatrix} \tag{2.7}$$

The mass squared eigenvalues of these matrices, $(m^2_{1/2})_i$ and $(m_0^2)_i$ respectively, are computed and substituted in the Coleman-Weinberg formula[8]

$$V_{\rm cw} = \frac{1}{64\pi^2}\left(\sum_i (m_0^2)_i^2 \ln(m_0^2)_i - 2\sum_i (m^2_{1/2})_i^2 \ln(m^2_{1/2})_i\right). \tag{2.8}$$

The effective potential is thus a function of the vevs of the background fields $\varphi_i$.

To check the proper functioning of the Coleman-Weinberg computation, `Vscape` allows to check the supertrace of the tree-level squared mass matrices. The supertrace is computed completely analogous to the Coleman-Weinberg potential, in that the eigenvalues of the matrices (2.6) and (2.7) are computed. The eigenvalues are then substituted in the formula:

$$\mathrm{STr}(M^2) = \sum_i (m_0^2)_i - 2\sum_i (m^2_{1/2})_i\,. \tag{2.9}$$

### 2.3 Metastable vacua

`Vscape` contains several commands to scan for local minima on the one-loop effective potential surface. Once a local minima of the effective potential is found, one can compute the one-loop corrected masses at the minima from:

$$V_{\rm eff}|_{\rm min} = V_{\rm eff}|_{\rm min} + \frac{1}{2}\, H_{ij}|_{\rm min}\, \varphi_i\varphi_j + \ldots \tag{2.10}$$

where $H_{ij}$ is the Hessian matrix. The eigenvalues of the Hessian are the mass squareds of the associated mass eigenstates. The mass squared eigenvalues allow you to verify (within error bounds, see section B.8) whether the minimum is indeed stable. Depending on the setup studied, the actual mass squared eigenvalues can be used as input parameters for SSM spectrum generators.

## 3. Structure of the program

The physics of the supersymmetric model is entirely encoded in the class CModel which is defined in the files `phenomodel.h` and `phenomodel.cc`. CModel has several routines relying on symbolic computations using the GiNaC library while other GSL-based routines are numerical.

The user-interface of `Vscape` was build on `ginsh`, the interactive frontend for the GiNaC symbolic computation framework [11]. This lightweight package for symbolic computations provided the appropriate user-interface to communicate between the phenomenological model and the user. We slightly extended the commands of the original `ginsh` application, to fit our needs.

`Ginsh` is implemented using the tools flex and bison which provide the detailed code for the lexer which reads the input (`ginsh_lexer.ll`) and parser which interprets the input (`ginsh_parser.yy`). `Ginsh` contains several commands, which are not related to the underlying phenomenological model. Those commands are detailed in Appendix A.

The original `ginsh` implementation allowed for additional commands to be added by other programmers in the separate file `ginsh_extensions.h`. The commands that specifically relate to the phenomenological model are defined in `vscape.h` and `vscape_functions.h`. They are included via `ginsh_extensions.h`. The shorter functions are implemented in `vscape.h` while the longer, mathematically involved algorithms such as minimization, are implemented in `vscape_functions.h`. Appendix B details the specific commands which allow you to interact with the phenomenological model CModel.

## 4. Outlook

This project grew out of a program written to study the Pentagon Model [13] [14]. Originally, the definition of the model was hardwired into the code. The added symbolic interface of `Vscape` gives the user the freedom to define the physics of the model her/himself. The commands available in `Vscape` provide the functionality that we needed to study the Pentagon Model. There are several additional features which might be interesting to include.

- Including vector multiplets in the computation of the Coleman-Weinberg potential.
- Allowing for a more general Kähler potential instead of the currently assumed canonical one.

- Specific commands to study the barrier heights and associated tunneling probabilities.
- Extending the user interface by introducing indexed symbols and control flow statements such as if-then statements and loops.
- Faster symbolic-numeric interface (technical extension). Currently the interface between the symbolic library GiNaC and the numeric library GSL is based on the GiNaC subs() command, which is rather time consuming. One could create a derived class of the GiNaC numeric class, which stores numerical data as actual C++ doubles. The implementation of the automatically invoked eval() command in the class should then take care of numeric evaluation, without having to do explicit subs() command. GSL could then manipulate directly on the doubles of this class.

## 5. Acknowledgements

I would like to thank my advisor T. Banks for his support and guidance throughout all the stages of this project. I am very grateful to R. Essig, K. Sinha, S. Thomas and G. Torroba for testing the program and for their suggestions. I also benefited from discussions with J. Jones, S. Lukic and J. Mason. A word of thanks to all the programmers contributing to impressive libraries that are freely available for everyone. I would like to thank SCIPP, where part of this work was done, for their hospitality. This work is supported by the Rutgers Department of Physics and Astronomy and DOE grant number DE-FG02-96ER40949.

## A. General Symbolic Commands

In this section, we detail the general math commands known to the Vscape. It is a slightly extended set of the original ginsh commands. The original manual of ginsh can be found as the linux ginsh manual.

### A.1 Running Vscape

You run Vscape by typing

```
./Vscape [file ...]
```

where the list of files is optional. Vscape will first attempt to read and interpret the commands from the file(s). When no files were specified or all the commands in the files have been executed, Vscape displays a prompt (`> `) signifying that it is ready to accept your input. Acceptable input are numeric or symbolic expressions consisting of numbers (e.g. `42`, `2/3` or `0.17`), symbols (e.g. `x` or `result`), mathematical operators like `+` and `*`, and functions (e.g. `sin` or `normal`). Every input expression must be terminated with either a semicolon (`;`) or a colon (`:`). If terminated with a semicolon, Vscape will evaluate the expression and print the result to stdout. If terminated with a colon, Vscape will only evaluate the expression but not print the result. It is possible to enter multiple expressions on one line. Whitespace (spaces,

tabs, newlines) can be applied freely between tokens. To quit Vscape, enter `quit` or `exit`, or type an EOF (Ctrl-D) at the prompt.

### A.2 Comments

Anything following a double slash (`//`) up to the end of the line, and all lines starting with a hash mark (`#`) are treated as a comment and ignored.

### A.3 Numbers

Vscape accepts numbers in the usual decimal notations. This includes arbitrary precision integers and rationals as well as floating point numbers in standard or scientific notation (e.g. `1.2E6`). The general rule is that if a number contains a decimal point (`.`), it is an (inexact) floating point number, otherwise it is an (exact) integer or rational. Integers can be specified in binary, octal, hexadecimal or arbitrary (2-36) base by prefixing them with `#b`, `#o`, `#x`, or `#nR`, respectively.

### A.4 Symbols

Symbols are made up of a string of alphanumeric characters and the underscore (`_`), with the first character being non-numeric. E.g. `a` and `mu_1` are acceptable symbol names, while `2pi` is not. It is possible to use symbols with the same names as functions (e.g. `sin`), Vscape is able to distinguish between the two.

Symbols can be assigned values by entering

*symbol* `=` *expression*

To unassign the value of an assigned symbol, type

`unassign('`*symbol*`');`

Assigned symbols are automatically replaced by their assigned value when they are used. To refer to the unevaluated symbol, put single quotes (`'`) around the name, as demonstrated for the `unassign` command above.

Symbols are considered to be in the complex or real domain depending on the mode in which they were defined. If the program is in the complex mode, every new symbol that has not been used before during the session is considered to be complex. If the program is in the real mode, every newly encountered symbol is considered to be real. The mode of the program only affects newly defined symbols. Thus when the program is in the real mode, symbols that were defined previously in the complex mode, remain complex. This mode of the program is switch using the keywords `real_symbols` and `complex_symbols`. The program is in the complex mode by default.

The following symbols are pre-defined constants that cannot be assigned a value by the user:

| | |
|---|---|
| `Pi` | Archimedes' Constant |
| `Catalan` | Catalan's Constant |
| `Euler` | Euler-Mascheroni Constant |
| `I` | imaginary unit $i$ |
| `FAIL` | an object of the GiNaC 'fail' class |

There is also the special

`Digits`

symbol that controls the numeric precision of calculations with inexact numbers. Assigning an integer value to digits will change the precision to the given number of decimal places.

### A.5 Wildcards

The `has()`, `find()`, `match()` and `subs()` functions accept wildcards as placeholders for expressions. These have the syntax

`$`*number*

for example `$0`, `$1` etc.

### A.6 Last printed expressions

Vscape provides the three special symbols

`%, %%` *and* `%%%`

that refer to the last, second last, and third last printed expression, respectively. These are handy if you want to use the results of previous computations in a new expression.

### A.7 Operators

Vscape provides the following operators, listed in falling order of precedence:

| | |
|---|---|
| `!` | postfix factorial |
| `^` | powering |
| `+` | unary plus |
| `-` | unary minus |
| `*` | multiplication |
| `/` | division |
| `+` | addition |
| `-` | subtraction |
| `<` | less than |
| `>` | greater than |
| `<=` | less or equal |
| `>=` | greater or equal |

`==` equal
`!=` not equal
`=` symbol assignment

All binary operators are left-associative, with the exception of `^` and `=` which are right-associative. The result of the assignment operator (`=`) is its right-hand side, so it's possible to assign multiple symbols in one expression (e.g. `a = b = c = 2;`).

### A.8 Lists

A list consists of an opening curly brace (`{`), a (possibly empty) comma-separated sequence of expressions, and a closing curly brace (`}`). A list is not a set in that it can contain several times the same element (e.g. `{1,2,3,4,3,3,x^2}`).

### A.9 Matrices

A matrix consists of an opening square bracket (`[`), a non-empty comma-separated sequence of matrix rows, and a closing square bracket (`]`). Each matrix row consists of an opening square bracket (`[`), a non-empty comma-separated sequence of expressions, and a closing square bracket (`]`). If the rows of a matrix are not of the same length, the width of the matrix becomes that of the longest row and shorter rows are filled up at the end with elements of value zero.

### A.10 Functions

A function call in Vscape has the form

*name*`(`*arguments*`)`

where *arguments* is a comma-separated sequence of expressions. Vscape provides a couple of built-in functions and also 'imports' all symbolic functions defined by GiNaC and additional libraries. There is no way to define your own functions other than linking Vscape against a library that defines symbolic GiNaC functions.

Vscape provides Tab-completion on function names: if you type the first part of a function name, hitting Tab will complete the name if possible. If the part you typed is not unique, hitting Tab again will display a list of matching functions. Hitting Tab twice at the prompt will display the list of all available functions.

A list of the built-in functions follows. They nearly all work as the respective GiNaC methods of the same name, so I will not describe them in detail here. Please refer to the GiNaC documentation.

`append(`*list1*`,` *list2*`)`
appends list2 to list1
`charpoly(`*matrix*`,` *symbol*`)`
characteristic polynomial of a matrix

`coeff(`*expression*`, `*object*`, `*number*`)`
extracts coefficient of *object^number* from a polynomial

`collect(`*expression*`, `*object-or-list*`)`
collects coefficients of like powers (result in recursive form)

`collect_distributed(`*expression*`, `*list*`)`
collects coefficients of like powers (result in distributed form)

`collect_common_factors(`*expression*`)`
collects common factors from the terms of sums

`conjugate(`*expression*`)`
complex conjugation

`content(`*expression*`, `*symbol*`)`
content part of a polynomial

`decomp_rational(`*expression*`, `*symbol*`)`
decompose rational function into polynomial and proper rational function

`degree(`*expression*`, `*object*`)`
degree of a polynomial

`denom(`*expression*`)`
denominator of a rational function

`determinant(`*matrix*`)`
determinant of a matrix

`diag(`*expression...*`)`
constructs diagonal matrix

`diff(`*expression*`, `*symbol*` [, `*number*`])`
partial differentiation

`divide(`*expression*`, `*expression*`)`
exact polynomial division

`eigenherm(`*matrix*`)`
compute the eigenvalues and eigenvectors of an n by n hermitian matrix M of numbers. The function returns a list with a 1 by n matrix of eigenvalues followed by the n by n unitary matrix were each column is an eigenvector. The columns of eigenvector are ordered in the same way the eigenvalues appear in the 1 by n matrix. Note: this is a numerical command, it does not accept expressions that cannot be evaluated numerically. The command works with double precision internally.

`eigensymm(`*matrix*`)`

compute the eigenvalues and eigenvectors of an n by n real symmetric matrix M of numbers. The function returns a list with a 1 by n matrix of eigenvalues followed by the n by n orthogonal matrix were each column is an eigenvector. The columns of eigenvector are ordered in the same way the eigenvalues appear in the 1 by n matrix. Note: this is a numerical command, it does not accept expressions that cannot be evaluated numerically. The command works with double precision internally.

`eval(`*expression* `[,` *level*`])`
evaluates an expression, replacing symbols by their assigned value

`evalf(`*expression* `[,` *level*`])`
evaluates an expression to a floating point number

`evalm(`*expression*`)`
evaluates sums, products and integer powers of matrices

`expand(`*expression*`)`
expands an expression

`find(`*expression*`,` *pattern*`)`
returns a list of all occurrences of a pattern in an expression

`fsolve(`*expression*`,` *symbol*`,` *number*`,` *number*`)`
numerically find rootof a real-valued function within an interval

`gcd(`*expression*`,` *expression*`)`
greatest common divisor

`has(`*expression*`,` *pattern*`)`
returns '1' if the first expression contains the pattern as a subexpression, '0' otherwise

`importfile(`*filename*`)`
read commands from *filename*. Note that this command cannot be nested in files.

`integer_content(`*expression*`)`
integer content f a polynomial

`inverse(`*matrix*`)`
inverse of a matrix

`is(`*relation*`)`
returns '1' if the relation is true, '0' otherwise (false or undecided)

`lcm(`*expression*`,` *expression*`)`
least common multiple

`lcoeff(`*expression*`,` *object*`)`
leading coefficient of a polynomial

`ldegree(`*expression*`,` *object*`)`
low degree of a polynomial

`lsolve(`*equation-list*`,` *symbol-list*`)`
solve system of linear equations

`map(`*expression*`,` *pattern*`)`

apply function to each operand, the function to be applied is specified as a pattern with the '$0' wildcard standing for all the operands

`match(`*expression*`,` *pattern*`)`

check whether expression matches a pattern

returns a list of wildcard substitutions or 'FAIL' if there is no match

`minus(`*list1*`,` *list2*`)`

returns the set-theoretic difference of list1 and list2

`nops(`*expression*`)`

number of operands in expression

`normal(`*expression* `[,` *level*`])`

rational function normalization

`numer(`*expression*`)`

numerator of a rational function

`numer_denom(`*expression*`)`

numerator and denumerator of a rational function as a list

`op(`*expression*`,` *number*`)`

extract operand *number* from *expression*, can also be used to get elements in a matrix

`power(`*expression1*`,` *expression2*`)`

exponentiation (equivalent to writing *expression1*`^`*expression2*)

`prem(`*expression*`,` *expression*`,` *symbol*`)`

pseudo-remainder of polynomials

`primpart(`*expression*`,` *symbol*`)`

primitive part of a polynomial

`quo(`*expression*`,` *expression*`,` *symbol*`)`

quotient of polynomials

`rank(`*matrix*`)`

rank of a matrix

`rem(`*expression*`,` *expression*`,` *symbol*`)`

remainder of polynomials

`resultant(`*expression*`,` *expression*`,` *symbol*`)`

resultant of two polynomials with respect to symbols

`rnd(`*number1*`,` *number2*`)`

generate a random number with double precision with uniform distribution within the interval [number1, number2]

`save(`*filename*`[,` *string*`] [,` *expression*`])`

open *filename* and append *string* and/or *expression*; to file

`series(`*expression*`,` *relation-or-symbol*`,` *order*`)`

series expansion

`sprem(`*expression*`,` *expression*`,` *symbol*`)`

sparse pseudo-remainder of polynomials

`sqrfree(`*expression* `[,` *symbol-list*`])`

square-free factorization of a polynomial

`sqrt(`*expression*`)`
square root

`symbolicmatrix(`*integer1*`,` *integer1*`,` *symbol*`)`
create an *integer1* by *integer2* matrix with entries *symbol_XX*

`subs(`*expression*`,` *relation-or-list*`)`

`subs(`*expression*`,` *look-for-list*`,` *replace-by-list*`)`
substitute subexpressions (you may use wildcards)

`tcoeff(`*expression*`,` *object*`)`
trailing coefficient of a polynomial

`time(`*expression*`)`
returns the time in seconds needed to evaluate the given expression

`trace(`*matrix*`)`
trace of a matrix

`transpose(`*matrix*`)`
transpose of a matrix

`unassign(`*symbol*`)`
unassign an assigned symbol

`unit(`*expression*`,` *symbol*`)`
unit part of a polynomial

`unique(`*list*`)`
removes all multiple occurrences of expressions in the list

### A.11 Special Commands

To exit Vscape, enter `quit` or `exit`. Vscape can display a (short) help for a given topic (mostly about functions and operators) by entering `?`*topic*. Typing `??` will display a list of available help topics. The command

`print(`*expression*`);`

will print a dump of GiNaC's internal representation for the given *expression*. This is useful for debugging and for learning about GiNaC internals. The command

`print_latex(`*expression*`);`

prints a LaTeX representation of the given *expression*. The command

`print_csrc(`*expression*`);`

prints the given *expression* in a way that can be used in a C or C++ program. The command

`iprint(`*expression*`);`

prints the given *expression* (which must evaluate to an integer) in decimal, octal, and hexadecimal representations. Finally, the shell escape

`![`*command* `[`*arguments*`]]`

passes the given *command* and optionally *arguments* to the shell for execution. With this method, you can execute shell commands from within Vscape without having to quit.

### A.12 Error messages

When you enter something which Vscape is unable to parse, it will report

`syntax error, unexpected` *foo* `at` *bar*

Please check the syntax of your input and try again. If the computer reports

`argument` *num* `to` *function* `must be a` *type*

it means that the argument number *num* to the given *function* has to be of a certain type (e.g. a symbol, or a list). The first argument has number 0, the second argument number 1, etc.

### A.13 Examples

```
> a = x^2-x-2;
-2-x+x^2
> b = (x+1)^2;
(x+1)^2
> s = a/b;
(1+x)^(-2)*(-2-x+x^2)
> diff(s, x);
-2*(1+x)^(-3)*(-2-x+x^2)+(-1+2*x)*(1+x)^(-2)
> normal(s);
(1+x)^(-1)*(-2+x)
> x = 3^50;
717897987691852588770249
> s;
717897987691852588770247/717897987691852588770250
> Digits = 40;
40
> evalf(s);
0.9999999999999999999999958211332927043849609906 79
> unassign('x');
x
> s;
(1+x)^(-2)*(-2-x+x^2)
>
> s = x^3 + x^2 + y + 1;
```

```
1+y+x^3+x^2
> map(s, $0^2);
1+y^2+x^4+x^6
> match(s, $0+$1^2);
{$1==x,$0==1+y+x^3}
>
> series(sin(x),x==0,6);
1*x+(-1/6)*x^3+1/120*x^5+Order(x^6)
>
> lsolve({3*x+5*y == 7}, {x, y});
{x==-5/3*y+7/3,y==y}
> lsolve({3*x+5*y == 7, -2*x+10*y == -5}, {x, y});
{x==19/8,y==-1/40}
>
> M = [ [a, b], [c, d] ];
[[-x+x^2-2,(x+1)^2],[c,d]]
> op(M, 0); op(M, 2);
-x+x^2-2
c
> determinant(M);
-2*d-2*x*c-x^2*c-x*d+x^2*d-c
> collect(%, x);
(-d-2*c)*x+(d-c)*x^2-2*d-c
>
> save("test.txt", "# test saving M to file");
Wrote to file <test.txt>: "# test saving M to file"
> save("test.txt", "M = ", M);
Wrote to file <test.txt>: "M = [[-2-x+x^2,(1+x)^2],[c,d]];"
> save("test.txt", M);
Wrote to file <test.txt>: "[[-2-x+x^2,(1+x)^2],[c,d]];"
>
> solve quantum field theory;
syntax error, unexpected T_SYMBOL at quantum
> quit
```

In this example, three lines where added to the end of the file `"test.txt"`:

```
# test saving M to file
M =[[-2-x+x^2,(1+x)^2],[c,d]];
[[-2-x+x^2,(1+x)^2],[c,d]];
```

If the file did not exist originally, it was created the first time the command `save()` was executed.

## B. Commands to find and analyse metastable vacua

In this section, we introduce the Vscape commands that allow you to interact with the build in supersymmetric model in Vscape.

### B.1 Defining the model

First you need to define the supersymmetric model that you want to study. The command

`ModelConstruct(`*Params*`,` *Fields*`,` *Vevs*`,` *W*`,` *V_D*`)`

allows you to specify the model. *Params* is a list of parameters that appear in the model (e.g. coupling constants). *Fields* is a list of expressions, every symbol in the list will be considered a field in your model. *Vevs* is a subset of *Fields*, it contains the fields which you allow to acquire a non-zero vacuum expectation value. The order in which the symbols appear in *Fields* determines the ordering of fields in the mass matrices and in the list of F-terms (see example later on). The expression *W*, specifies the superpotential and *V_D* is the expression for the D-term potential which is added to the tree level potential. Both expressions may only contain symbols that are listed either in *Fields* or *Params*. The command `ModelConstruct()` will return `0` if it successfully defined the model, otherwise it will show an error message. Once the model is successfully loaded, you can use all the other commands described below to study the model.

The command `ModelConstruct()` computes the F-terms and mass-matrices symbolically. Depending on the number of fields in the model and calculational speed this command might take some time. Once the model is computed with `ModelConstruct()`, you can use the command

`ModelSave(`*filename*`)`

to save the computed model to *filename*. This command saves all the information about the model including F-terms and mass matrices such that, later on, when you want to load that specific model again, the computer does not have to redo the lengthy computation of the F-terms and mass matrices. `ModelSave()` will return `0` if it successfully saved the model otherwise it will show an error message.

Reloading a specific model, is done by simply invoking `importfile(`*filename*`)`. `Vscape` will execute the commands in *filename*. *filename* was created by `ModelSave()` such that it will first define the fields, parameters, vevs, superpotential, D-term potential, F-terms and mass matrices. The last line of *filename* will call the command `ModelLoad()` (discussed below) which does the actually loading of the model into `Vscape`. So after invoking `importfile(`*filename*`)`, where *filename* is a file that was created with `ModelSave()`, the model is loaded into `Vscape` and you can use all the commands described below to study the model.

I would advise against manually editing files created by `ModelSave()`, since they contain all the information of the model you defined with `ModelConstruct()` in a consistent way. It

contains the exact mass matrices and F-terms corresponding to the superpotential with the correct ordering. E.g. the order in which the F-terms appear in *filename* is determined by the order in which the fields are listed in *filename*.

The command

`ModelLoad(`*Params*`,` *Fields*`,` *Vevs*`,` *W*`,` *V_D*`,` *Fterms*`,` *mF*`,` *mB*`)`

is called implicitly as the last line of any file which was created with `ModelSave()`, so you will rarely need to use it explicitly. Like `ModelConstruct()`, this command loads a specific model into `Vscape` such that it can be studied with subsequent commands. However, in this case the F-terms and mass matrices need to be specified as an argument, so this command avoids the sometimes time-consuming computation of the F-terms and mass matrices. *Fterms* should be a list of expressions for the different F-terms. *mF* and *mB* are the fermion and scalar mass matrices respectively. The entries in *Fterms*, *mF* and *mB* should be ordered as the fields are ordered in *Fields*. Thus e.g. the second entry in the list *Fterms* is the F-term with respect to the second field appearing in the list *Fields*. Warning: the program does not check whether the given F-terms and mass matrices actually correspond to the superpotential given!

If you invoke `ModelConstruct()` (or implicitly `ModelLoad()` via `importfile()`), when a model was already loaded, the internal model of Vscape will be changed to the new model. Thus, all the subsequent commands will involve that new model.

**Example** This example and the subsequent examples use the model discussed in [2] with $N = 1$ and $N_f = 6$. A slightly modified version of this example is included in the distribution of Vscape and can be loaded with `importfile("`*path*`/examples/iss.txt")` where *path* is the correct path to the file. Note: the file `iss.txt` was not created with `ModelSave()`, the file does not contain F-terms nor mass matrices, so you savely modify it.

```
> ## Switch to real mode, to introduce two real parameters
> real_symbols;
  All new symbols introduced from now on are assumed to be real.
> Lambda:
> m_ISS:
>
> ## Switch to complex mode, to introduce all the other symbols
> complex_symbols;
  All new symbols introduced from now on are assumed to be complex.
>
> ## Define the model
> Params = {Lambda, m_ISS}:
> B  = symbolicmatrix(6,1,B_);
[[B_0],[B_1],[B_2],[B_3],[B_4],[B_5]]
> Bt = symbolicmatrix(1,6,Bt_):
> M  = symbolicmatrix(6,6,M_):
```

```
> Fields = {B,Bt,M}:
> Vevs   = {B_5,Bt_5,M_00,M_11,M_22,M_33,M_44,M_55}:
> W   = evalm(Bt*M*B)                //
>       - determinant(M)*Lambda^(-3) //
>       + Lambda*m_ISS*trace(M):
> V_D = 0:
>
> ## Construct the model
> ModelConstruct(Params, Fields, Vevs, W, V_D);
Loading Model...
  Computing F-terms and mass matrices...
  F-terms:       i = 047;
  Mass matrices: i = 047; j = 047; k = 047;
  Done.
Model successfully loaded...
0
>
> ## Now we save the model to a file
> ModelSave("examples/isscomputed.txt");
The internal model was successfully saved to <examples/isscomputed.txt>.
0
```

From now on this specific model can quickly be loaded using the command `importfile("examples/isscomputed.txt");`. Note that the file `isscomputed.txt` was created using `ModelSave()` and thus contains the correctly ordered F-terms and mass matrices of this specific model, so this file should not be modified by hand. Whenever you want to define a new model different from the above one (e.g. with one more term in the superpotential) you will have to use the command `ModelConstruct()` again)

### B.2 Symbolic commands for the model

There are several commands which give you symbolic information on the model that is loaded.

`ModelParams()`
: Returns the parameters of the model in a list

`ModelFields()`
: Returns the fields of the model in a list

`ModelVevs()`
: Returns the fields with non-zero expectation value in a list

`ModelW()`
: Returns the superpotential

`ModelVD()`
: Returns the D-term potential

`ModelV0()`

Returns the tree level potential symbolically

`ModelFtermsi(`*integer*`)`

Returns the F-terms related to field *integer*

`ModelFterms()`

Returns the F-terms in a list

`ModelmFij(`*a*`, `*b*`)`

Returns entry *a*, *b* of the fermion mass matrix: $W\_ab$

`ModelmBij(`*a*`, `*b*`)`

Returns entry *a*, *b* of the upper right block of the scalar mass matrix: $W * \_abc\ W\_c$

`ModelmF()`

Returns the fermion mass matrix: $W\_ab$

`ModelmB()`

Returns the upper right (off diagonal) block of the scalar mass matrix: $W * \_abc\ W\_c$

The list of F-terms and the entries in the mass matrices are ordered according to the occurence of the fields in `ModelFields()`. The *integer* values in the above command also refer to the occurence of the field in `ModelFields()`, the first field has integer `0`.

**Example** To try the example below, you will need the file `examples/isscomputed.txt` which was created in the example above.

```
> importfile("examples/isscomputed.txt");
> ModelFields();
{B_0,B_1,B_2,B_3,B_4,B_5,
 Bt_0,Bt_1,Bt_2,Bt_3,Bt_4,Bt_5,
 M_00,M_01,M_02,M_03,M_04,M_05,
 M_10,M_11,M_12,M_13,M_14,M_15,
 M_20,M_21,M_22,M_23,M_24,M_25,
 M_30,M_31,M_32,M_33,M_34,M_35,
 M_40,M_41,M_42,M_43,M_44,M_45,
 M_50,M_51,M_52,M_53,M_54,M_55}
> ModelFterms();
{0,0,0,0,0,Bt_5*M_55,
 0,0,0,0,0,M_55*B_5,
 m_ISS*Lambda-M_33*Lambda^(-3)*M_44*M_11*M_55*M_22,0,0,0,0,0,
 0,m_ISS*Lambda-M_33*M_00*Lambda^(-3)*M_44*M_55*M_22,0,0,0,0,
 0,0,-M_33*M_00*Lambda^(-3)*M_44*M_11*M_55+m_ISS*Lambda,0,0,0,
 0,0,0,-M_00*Lambda^(-3)*M_44*M_11*M_55*M_22+m_ISS*Lambda,0,0,
 0,0,0,0,m_ISS*Lambda-M_33*M_00*Lambda^(-3)*M_11*M_55*M_22,0,
 0,0,0,0,0,-M_33*M_00*Lambda^(-3)*M_44*M_11*M_22+Bt_5*B_5+m_ISS*Lambda}
>
```

```
> ## F-term wrt B_5;
> myFterm = ModelFtermsi(5);
Bt_5*M_55
>
> ## entry associated with B_0, Bt_0 in mass matrices;
> ModelmFij(0,6);
M_00
> ModelmBij(0,6);
m_ISS*Lambda-M_33*Lambda^(-3)*M_44*M_11*M_55*M_22
```

### B.3 Numerical commands for the model: the effective potential

All commands that involve numeric computations work with double precision, independent from the `Digits` setting for the symbolic computations. The commands to evaluate the potential numerically are,

`ModelnumV0(`*SubsList*`)`
: Evaluates the tree level potential numerically

`ModelnumVcw(`*SubsList*`)`
: Evaluates the Coleman-Weinberg potential numerically

`ModelnumVeff(`*SubsList*`)`
: Evaluates the one loop effective potentiel potential numerically

`ModelnumSTrM2(`*SubsList*`)`
: Evaluates the supertrace over scalar and fermion mass matrices

The *SubsList* is a list of equations which specify the value of the parameters and fields which were allowed to have non-zero expectation value.

**Example** The example below uses the file `examples/isscomputed.txt` which was created in an example above.

```
> importfile("examples/isscomputed.txt");
> Subs = {Lambda == 1000.0, m_ISS == 10.0,  //
>         B_5  == 0.00 + 100.0*I,           //
>         Bt_5 == 0.01 + 100.0*I,           //
>         M_00 == 0.0,                      //
>         M_11 == 0.0,                      //
>         M_22 == 0.0,                      //
>         M_33 == 0.0,                      //
>         M_44 == 0.0,                      //
>         M_55 == 0.0}:
> ModelnumVcw(Subs);
3.355198673876628 6522E7
```

```
> ModelnumSTrM2(Subs);
3.456079866737127304E-11
```

### B.4 Minima of the effective potential

You can find local minima of the one loop effective potential by using the command:

`ModelMinSimpl(`*SubsPattern*`,` *Variables*`,` *StartPoint*`)`

*SubsPattern* is a list of equations which specifies the values of the parameters and fields with non-zero vev as function of the variables. The variables are specified in the field *Variables* as a list of real symbols. The minimization algorithm will minimize the effective potential as a function of those variables. The field *StartPoint* is a list of initial value for each of the variables. The command returns a list with values for the variables at the minima. The value for simplex size shown while the command is running is discussed in section B.9.1

This command is calculationally intensive, the calculation time depends on the number of fields in the model. While the calculation progresses the computer will display the current iteration of the algorithm and the minimum value of the potential that has been reached so far. The algorithm used to minimize the function is the simplex algorithm of Nelder and Mead [15]. This algorithm has the advantage that it does not rely on computational intensive gradients to iterate towards the minimum. Some more details and parameters that influence the simplex algorithm are given in section B.9.1.

Another command that analyzes the effective potential is

`ModelStatBox(`*SubsPattern*`,` *Variables*`,` *StartPoint*`,` *Point1*`,` *Point2*`)`

where the fields *SubsPattern* and *Variables* have the same function described above. The command starts from the value of the effective potential where the variables have the value specified in *StartPoint*. The algorithm then picks several random points in the box defined by *Point1* and *Point2* and checks whether the value of the effective potential at that point is lower than the minimum value found so far. The command returns a list of values for the variables at the random point with lowest effective potential.

**Example** The example below uses the file `examples/isscomputed.txt` which was created in an example above.

```
> importfile("examples/isscomputed.txt");
> Subs = {Lambda == 1000.0, m_ISS == 10.0, //
>         B_5  == 0 + b*I,                 //
>         Bt_5 == c + d*I,                 //
>         M_00 == 0.0,                     //
>         M_11 == 0.0,                     //
>         M_22 == 0.0,                     //
>         M_33 == 0.0,                     //
```

```
>         M_44 == 0.0,                      //
>         M_55 == e}:
> ModelMinSimpl(Subs, {b,c,d,e}, {100,0,100,0});
       250:  Veff = +5.334976441748862266 54e+08
             Simplex size = +1.569691689254765876 82e-05
{98.755419109029958236, 8.655457583820324883 4E-7,
 98.755388956674764245, 7.920961955373477163 3E-7}
> ModelMinSimpl(Subs, {b,c,d,e}, {0,900,900,900});
       680:  Veff = +5.334976441748861074 45e+08
             Simplex size = +3.023429275407894359 54e-06
{98.7554136717079416,  -3.535338356861064858 6E-8,
 98.755391871241812396, 5.732497568898521597 6E-8}
>
>
> Subs = {Lambda == 1000.0, m_ISS == 10.0, //
>         B_5  == 0 + b*I,                  //
>         Bt_5 == c + d*I,                  //
>         M_00 == a,                        //
>         M_11 == a,                        //
>         M_22 == a,                        //
>         M_33 == a,                        //
>         M_44 == a,                        //
>         M_55 == e}:
> ModelMinSimpl(Subs,{a,b,c,d,e},{900,0,0,0,900});
       660:  Veff = -1.035094323781734476 49e-05
             Simplex size = +1.695415455893741844 33e-13
{398.10717886889341344,
   5.742825702433017801 3E-7,
   1.761789758548344484 2E-6,
  -8.394795160100672925 5E-7,
 398.107133987191105}
>
> ModelStatBox(Subs,{a,b,c,d,e},{0,98,0,98,0}, //
>              {200,0,0,0,200},{500,0,0,0,500});
   Checking random point no.    90; (so far     6 lower points found)
{381.46924413740634918, 0.0, 0.0, 0.0, 467.38906654063612223}
```

The first two minimizations started at different points but led to the same metastable minimum discussed in [2]. Both results are equal up to numerical precision as discussed in section B.8.2. The $U(1)_B$ symmetry is spontaneously broken giving rise to a Goldstone direction. We fixed this freedom by choosing $\mathrm{Re}B_5 = 0$.

The third minimization led to the supersymmetric vacua of the model. Notice that the rank-condition mechanism for supersymmetry breaking for models with $N = 1$ only holds around the origin in fieldspace. Away from the origin there is a point where all the F-terms become zero due to the additional determinant term in the superpotential. The numerical value for the effective potential at the supersymmetric point is slightly negative but zero within error bounds as discussed in section B.8.1.

The fourth operation was a statistical search. We started from a point close to the metastable vacuum and search randomly in the interval $M_{00} = \cdots = M_{44} \in [200, 500], M_{55} \in [200, 500]$. The algorithm returned a point in fieldspace relatively close to the supersymmetric minima with a lower value for the effective potential.

### B.5 Analyzing the vacua

The commands to analyse a minimum all use the same structure of arguments as described above: *SubsPattern* is a list of equations linking the parameters and fields with non-zero vev to the variables which are specified in the list *Variables*. The values for the variables are given in the list of numbers *Point*.

`ModelGradient(`*SubsPattern*`,` *Variables*`,` *Point*`)`
: Evaluates the gradient of the effective potential at *Point*. The command returns a list with the gradient followed by error flags on the gradient each formatted as a 1 by n matrix. The computer will show the estimated time to complete the algorithm since this computation is calculationally intensive.

`ModelHessian(`*SubsPattern*`,` *Variables*`,` *Point*`)`
: Evaluates the Hessian matrix at *Point*. The command returns a list with the hessian matrix followed by the matrix of error flags. The computer will show the estimated time to complete the algorithm since this computation is calculationally intensive.

`Modeldiff2(`*SubsPattern*`,` *Variables*`,` *Point*`,` *i*`,` *j*`,` *dx_i*`,` *dx_j*`)`
: Evaluates the second order derivative with respect to fields *i* and *j*. *dx_i* and *dx_j* are suggested stepsize to compute the derivative (see section B.9.3 for more information). The command returns a list with the second order derivative followed by the error flag.

`ModelNegEig(`*SubsPattern*`,` *Variables*`,` *Point*`,` *Hessian*`,` *HessianError*`)`
: Checks whether there are negative eigenvalues of the Hessian matrix of the effective potential at *Point*. You need to make sure that *Hessian* (and *HessianError*) are indeed the Hessian matrix (and error flags) of the effective potential at *Point*. The function will return the original *Point* or, if there are negative eigenvalues that are larger than the average errorflag, another *Point* where the effective potential has a lower value. Used as cross check for the other minimization commands.

**Example** The example below uses the file `examples/isscomputed.txt` which was created in an example above.

```
> importfile("examples/isscomputed.txt");
> Subs = {Lambda == 1000.0, m_ISS == 10.0, //
>         B_5  == 0.0 + b*I,                //
>         Bt_5 == 0.0 + d*I,                //
>         M_00 == 0.0,                      //
>         M_11 == 0.0,                      //
>         M_22 == 0.0,                      //
>         M_33 == 0.0,                      //
>         M_44 == 0.0,                      //
>         M_55 == 0.0}:
> Min = ModelMinSimpl(Subs, {b,d}, {100,100});
       160:  Veff = +5.33497644174886047840e+08
             Simplex size = +3.29244092296298119244e-06
{98.75539482502895794,98.75540931029662772}
>
>
> ModelGradient(Subs, {b,d}, {100,100});
      2/2:  00:01s to go...
{[[44055.476986700297857, 44055.476986670022598]],
 [[6.273248519121328748,      6.27324852896071139]]}
> ModelGradient(Subs, {b,d}, Min);
      2/2:  00:01s to go...
{[[-5.261715075073619019,    -5.25428542596663739 06]],
 [[ 3.8105577427291033032,    3.810557118465678 35]]}
>
> ModelHessian(Subs, {b,d}, Min);
      3/3:  00:19s to go...
{[[17652.30720837064655,     17140.548392404118204],
  [17140.548392404118204,    17652.301781115671474]],
 [[   10.064568391628851529,    10.455072325682765566],
  [   10.455072325682765566,    10.064565316884026558]]}
> Hessian = op(%, 0): HessianErr = op(%%, 1):
> eigensymm(Hessian);
{[[34792.852887147491856,      511.75610233882599687]],
 [[    0.7071068371597952762,   -0.7071067252132953173 7],
  [    0.7071067252132953173,    0.7071068371597952762]]}
```

The second column in the last matrix in the example is the eigenvector corresponding to the pseudo-moduli direction in [2]. The corresponding mass squared eigenvalue indeed corresponds to the theoretical value computed in [2]: $511.756\,\mathrm{GeV}^2$ versus $489.248\,\mathrm{GeV}^2$.

### B.6 Plots

There are three commands in Vscape to plot:

`ModelPlotV0(`*filename*`,` *SubsPattern*`,` *Variables*`,` *xmin*`,` *xmax*`,` *ymin*`,` *ymax*`)`
`ModelPlotVcw(`*filename*`,` *SubsPattern*`,` *Variables*`,` *xmin*`,` *xmax*`,` *ymin*`,` *ymax*`)`
`ModelPlotVeff(`*filename*`,` *SubsPattern*`,` *Variables*`,` *xmin*`,` *xmax*`,` *ymin*`,` *ymax*`)`

Each commands writes 3d plotting data of respectively $V_0$, $V_{\rm cw}$ or $V_{\rm eff}$ to the file *filename*. *SubsPattern* again indicates how the parameters and fields with non-zero vev depend on the *Variables*. *Variables* should be a list of 2 variables which will be on the x and y axis of the 3d plot. The variables will be evaluated between *xmin* and *xmax* (*ymin*, *ymax* resp.) with a stepsize of 1. The computer will show the progress for all plotting commands since it is calculationally intensive. The command returns 0 when it was successful. The file is formatted such that it can then easily be imported in other math packages. A maple file (`Plotting.mw`) which can import the data is included in the distribution of Vscape in the directory `examples/`. The file format is

```
xmin xmax
ymin ymax
f(xmin,ymin)   f(xmin+1,ymin)   ... f(xmax,ymin)
f(xmin,ymin+1) f(xmin+1,ymin+1) ... f(xmax,ymin+1)
...
f(xmin,ymax)   f(xmin+1,ymax)   ... f(xmax,ymax)
```

where the first two lines contain integer values, while all other data consists of doubles.

**Example** The example below uses the file `examples/isscomputed.txt` which was created in an example above.

```
> importfile("examples/isscomputed.txt");
> pi = evalf(Pi):
> M  = 398.1*b/80:
> B  = 98.755*(1 - b/80):
> Subs = {Lambda == 1000.0, m_ISS == 10.0,  //
>        B_5  == exp( 2*pi*a*I/10)*(B*I),   //
>        Bt_5 == exp(-2*pi*a*I/10)*(B*I),   //
>        M_00 == M,                         //
>        M_11 == M,                         //
>        M_22 == M,                         //
>        M_33 == M,                         //
>        M_44 == M,                         //
>        M_55 == M}:
> ModelPlotVeff("BarrierPlot.txt", Subs, {a,b}, 0, 9, 0,99);
```

```
The 3d plotting data was successfully saved to <BarrierPlot.txt>.
0
```

The plot obtained from the example using `examples/plotting.mw` in Maple 10 is shown in figure 1.

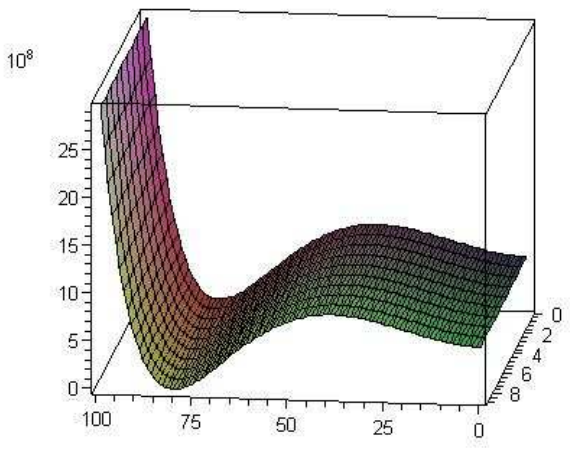


**Figure 1:** The effective potential from the metastable vacuum to the supersymmetry vacuum. The axis into the paper is the flat Goldstone direction of the $U(1)_B$. The axis along the paper cuts through the one-loop effective potential from the metastable minima on the left to the supersymmetric vacuum on the right side.

### B.7 Interface with spectrum generators

The command

`ModelSpectr(`*Filename*`,` *SubsList*`)`

allows interaction with spectrum generators. The command takes the file *filename* as input file. This file should be formatted according to the Les Houches Accord [16], except that numbers can be replaced by formulas. `ModelSpectr()` will substitute the values specified in *SubsList* and the current values of all other variables used up to that point in Vscape and generate an outputfile `<filename.out>`. This file can then be used as input file for spectrum generators. Whenever an unknown symbol is encountered, the entire line is copied verbatim without substitution and a message is printed on the screen. Vscape will add a Block containing version information to the output file, to indicate that the file was generated with Vscape.

**Example** The example below uses the file `examples/isscomputed.txt` which was created in an example above. As input file in this example, we use `LesHoucheSymbolic.txt`

```
# SUSY Les Houches Accord 1.0 - example input file
#
Block MODSEL                # Select Model
     1    2                 #   (m)GMSB
Block SMINPUTS              # Standard Model inputs
     3    0.1172            #   alpha_s(MZ) SM MSbar
Block MINPAR                # Susy breaking input parameters
     1  sqrt(Lambda*m_ISS)  #   scale of soft susy breaking
     3  ReHu0/ReHd0         #   tanb
     4    1.0               #   sign(mu)
Block EXPAR                 # Non-minimal susy breaking input parameters
     23   g_S*S             #   mu-term
```

The commands sequence of the example is

```
> importfile("examples/isscomputed.txt");
> Subs = {Lambda == 1000.0, m_ISS == 10.0, //
>         B_5  == 0 + b*I,                 //
>         Bt_5 == c + d*I,                 //
>         M_00 == 0.0,                     //
>         M_11 == 0.0,                     //
>         M_22 == 0.0,                     //
>         M_33 == 0.0,                     //
>         M_44 == 0.0,                     //
>         M_55 == e}:
> ModelMinSimpl(Subs, {b,c,d,e}, {100,0,100,0});
       250:  Veff = +5.334976441748862266 54e+08
             Simplex size = +1.5696916892547658768 2e-05
{98.755419109029958236, 8.6555675838203248834E-7,
 98.755388956674764245, 7.9209619553734771633E-7}
> MinSubs = subs(Subs, {b,c,d,e}, %);
{Lambda==1000.0, m_ISS==10.0,
 B_5==98.755419109029958236*I,
 Bt_5==8.6555675838203248834E-7+98.755388956674764245*I,
 M_00==0.0, M_11==0.0, M_22==0.0, M_33==0.0, M_44==0.0,
 M_55==7.9209619553734771633E-7}
> g_S = 0.6:
> S   = 0.001:
> ModelSpectr("LesHouchesSymbolic.txt", MinSubs);
```

```
unknown symbol 'ReHu0'
 => just copied this data line from inputfile to outputfile...
Unable to match all symbols in <LesHouchesSymbolic.txt>
with currently defined symbols...
2
```

This example produces the output file `<LesHouchesSymbolic.txt.out>`

```
# SUSY Les Houches Accord 1.0 - example input file
#
Block VSCAPE          # program info
     1     Vscape     # program
     2     1.0.0      # version number
Block MODSEL                # Select Model
     1   2                      #   (m)GMSB
Block SMINPUTS              # Standard Model inputs
     3   0.1172                 #   alpha_s(MZ) SM MSbar
Block MINPAR                # Susy breaking input parameters
     1   100.0       #   scale of soft susy breaking
     3  ReHu0/ReHd0         #   tanb
     4   1.0                    #   sign(mu)
Block EXPAR                 # Non-minimal susy breaking input parameters
     23   6.0E-4                  #   mu-term
```

Notice that the line specifying $\tan\beta$ was not replaced by a number, since our simple example did not specify a value for $\mathrm{Re}H_u^0$.

### B.8 Numerical precision and errorflags

The numerical format used for all the numerical computations in Vscape is double precision, which has 52 significant bits or approximately 16 decimal digits.

#### B.8.1 Precision of the Coleman-Weinberg potential

In one of the examples above, we found that the effective potential at the supersymmetric minima had a negative numerical value of order $-10^{-5}$. The value for the vevs of some of the fields at that point was around 400. This implies that some of the squares of the mass squared eigenvalues of the mass matrices will be of the order $m^4 \approx 400^4 \approx 10^{11}$. Those squares of mass squared eigenvalues appear in the Coleman-Weinberg potential. As the internal numerical algorithm uses double precision, we can expect the absolute error on the potential to be estimated by

$$|\phi_{\max}|^4 10^{-16}\,. \tag{B.1}$$

Or, in our example of the order of $10^{-5}$, which implies that the value for the effective potential at the supersymmetric minimum is zero within error bounds.

### B.8.2 Precision of the vevs after minimization

In another example, we found two slightly different sets of vevs for the fields at the metastable minima, since we started the minimization process from a different point. The relative difference between the vevs was of order $10^{-7}$. The vevs of some of the fields at the metastable minima was of order $10^2$. Following the reasoning from the previous paragraph, we estimate that the absolute error on the effective potential is of order $10^{-8}$ and since the actual value of the effective potential at the minimum was of order $10^8$, the relative error is of order $10^{-16}$. Minimizing the potential then holds,

$$V_{\text{eff}}(\phi_i) \approx V_{\text{eff}}(\phi_i) \pm |\phi_{\max}|^4 10^{-16} \tag{B.2}$$

$$= V_{\text{eff}}(\phi_i)(1 \pm 10^{-16}) \tag{B.3}$$

$$= V_{\text{eff}}(\phi_i + \epsilon_i) \tag{B.4}$$

$$= V_{\text{eff}}(\phi_i) + \frac{1}{2}\frac{\partial^2 V_{\text{eff}}}{\partial\phi_i \partial\phi_j}\epsilon_i \epsilon_j \tag{B.5}$$

where $\epsilon_i$ is the error on the vev of the field $\phi_i$ at the minimum coming from the limited precision on $V_{\text{eff}}$. In the example above, $V_{\text{eff}}$ was of order $10^8$ at the metastable minima, while the second order derivative can be at most of order the cutoff squared, which was $10^6$. We find that the highest precision on the fields at the metastable minimum is of order $10^{-7}$, which corresponds to the results found in the example. A lower value for the second order derivative gives less precision.

## B.9 Parameters controlling the internal algorithms

### B.9.1 Simplex minimization

The algorithm that is used for the command `ModelMinSimpl()` is the Simplex algorithm by Nelder and Mead [15]. An n-simplex is chosen around the startpoint in variable space. By changing the vertices of the simplex, the algorithm minimizes the size of the simplex step by step. When the size of the simplex is smaller than *MinimizerSizeTest*, the algorithm considers the minimum to be found. When the command `ModelMinSimpl()` is running, the current size of the simplex is shown on the screen. The parameter *MinimizerInitialStep* determines the size of the initial simplex with which the algorithm starts. *MinimizerMaxIterations* sets the maximal number of iterations, the algorithm will quit even if the algorithm has not found a minimum. Sometimes the size of the simplex never becomes small enough to get a succesfull end of the algorithm, while the simplex is already at its most optimal position. The parameter *MinimizerMaxDeadIterations* determines the maximal number of iterations that the algorithm can proceed without finding a point with a lower functional value.

### B.9.2 Statistical minimization

The number of random points the command `ModelStatBox()` test for a lower value of the effective potential, is determined by the parameter *StatBoxMaxIterations*.

### B.9.3 Derivatives

The numerical derivatives are computed using the 11-point rule:

$$\begin{aligned}\frac{df(x_0)}{dx} \approx \frac{1}{2520\Delta x}\Big( & -2f(x_0-5\Delta x)+25f(x_0-4\Delta x)-150f(x_0-3\Delta x)\\ & +600f(x_0-2\Delta x)-2100f(x_0-1\Delta x)\\ & +2100f(x_0+1\Delta x)-600f(x_0+2\Delta x)\\ & +150f(x_0+3\Delta x)-25f(x_0+4\Delta x)+2f(x_0+5\Delta x)\Big) \end{aligned} \quad \text{(B.6)}$$

where $\Delta x$ is the stepsize of the algorithm. The algorithm estimates the errorflags coming from rounding errors where the relative error on the effective potential is set by *diffErrorOnf*. In addition, the truncation error which is estimated by comparing the 9-point rule with the 11-point rule result is also determined. The rounding error scales as $\Delta x^{-1}$ while the truncation error goes like $\Delta x^8$. Using those two competing errors, the algorithm estimates a new optimal $\Delta x$ and repeats the process until the relative error is smaller than *diffTolerance* [17]. It will iterate at most *diffMaxCycle* times. One more parameter determines the behaviour of the algorithm, if the new guess for an optimal $\Delta x$ is bigger than *diffMaxdx* then the algorithm quits. The command `Modeldiff2()` allows you to suggest the initial stepsizes of the derivatives in the $i$ and $j$ direction with the arguments *dx_i*, *dx_j*. The other commands make an automatic initial estimate.

### B.9.4 Changing the control parameters

You can get the values of all the parameters discussed above with the command

`GetPrecision()`

You can set the parameters with the command

`SetPrecision(`*StatBoxMaxIterations*`,` *MinimizerSizeTest*`,` *MinimizerInitialStep*`,` *MinimizerMaxIterations*`,` *MinimizerMaxDeadIterations*`,` *diffMaxCycle*`,` *diffTolerance*`,` *diffMaxdx*`,` *diffErrorOnf*`)`

If you set an argument in `SetPrecision()` to zero, the parameter will maintain its old value.

The default parameters of the program have been optimized for models of the kind used in the examples, with a cutoff of the order of 1000 where we think in units of GeV. Different models might require different parameters. For example, if one has a very shallow metastable minima, *MinimizerInitialStep* should not be chosen too large, since this would result in the algorithm jumping out of the metastable minima in favour of another minima with a lower effective potential.

## References


[1] E. Witten, *Dynamical Breaking of Supersymmetry*, *Nucl. Phys.* **B 188** (1981) 513.
K. Intriligator, N. Seiberg, *Lectures on Supersymmetry Breaking*, `hep-ph/0702069`.

[2] K. Intriligator, N. Seiberg, D. Shih, *Dynamical SUSY breaking in meta-stable vacua*, *J. High Energy Phys.* **04** (2006) 021, hep-th/0602239.

[3] M. Dine, J. L. Feng, E. Silverstein, *Retrofitting O'Raifeartaigh models with dynamical scales*, *Phys. Rev.* **D 74** (2006) 095012, hep-th/0608159.

[4] M. Dine, J. Mason, *Gauge mediation in metastable vacua*, hep-ph/0611312.

[5] D. Shih, *Spontaneous R-symmetry breaking in O'Raifeartaigh models*, hep-th/0703196.
K. Intriligator, N. Seiberg, D. Shih, *Supersymmetry breaking, R-symmetry breaking and metastable vacua*, hep-th/0703281.

[6] F. Denef, M. R. Douglas, *Distributions of flux vacua*, *J. High Energy Phys.* **05** (2004) 072, hep-th/0404116.
M. Dine, D. O'Neil, Z. Sun, *Branches of the Landscape*, *J. High Energy Phys.* **07** (2005) 014, hep-th/0501214.
M. R. Douglas, J. Shelton, G. Torroba, *Warping and Supersymmetry Breaking*, arXiv:0704.4001 [hep-th].

[7] G. F. Giudice, R. Rattazzi, *Theories with gauge mediated supersymmetry breaking*, *Phys. Rept.* **322** (1999) 419, hep-ph/9801271.
Y. Shadmi, Y. Shirman, *Dynamical supersymmetry breaking*, *Rev. Mod. Phys.* **72** (2000) 25, hep-th/9907225.
M. A. Luty, *2004 TASI lectures on supersymmetry breaking*, hep-th/0509029.

[8] S. Coleman, E. Weinberg, *Radiative Corrections as the Origin of Spontaneous Symmetry Breaking*, *Phys. Rev.* **D 7** (1973) 1888.

[9] B. C. Allanach, *SOFTSUSY: A C++ program for calculating supersymmetric spectra*, *Comput. Phys. Commun.* **143** (2002) 305, hep-ph/0104145.

[10] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, F. Rossi, *GNU Scientific Library Reference Manual - Revised Second Edition*, ISBN 0954161734.

[11] C. Bauer, A. Frink, R. Kreckel, *Introduction to the GiNaC Framework for Symbolic Computation within the C++ Programming Language*, J. Symbolic Computation **33** (2002) 1.
J. Vollinga, *GiNaC: Symbolic computation with C++*, *Nucl. Instrum. Meth.* **A559** (2006) 282, hep-ph/0510057.

[12] S. Martin, *A Supersymmetry Primer*, hep-ph/9709356.

[13] T. Banks, *Remodeling the Pentagon After the Events of 2/23/06*, hep-ph/0606313.

[14] T. Banks, K. van den Broek, to appear.

[15] J. A. Nelder, R. Mead, *A Simplex Method for Function Minimization*, Comput. J. **7** (1965) 308.

[16] P. Skands, B. C. Allanach, H. Baer, C. Balázs, G. Bélanger, F. Boudjema, A. Djouadi, R. Godbole, J. Guasch, S. Heinemeyer, W. Kilian, J-L. Kneur, S. Kraml, F. Moortgat, S. Moretti, M. Mühlleitner, W. Porod, A. Pukhov, P. Richardson, S. Schumann, P. Slavich, M. Spira, G. Weiglein, *SUSY Les Houches Accord: Interfacing SUSY Spectrum Calculators, Decay Packages, and Event Generators*, *J. High Energy Phys.* **07** (2004) 036, hep-ph/0311123.

[17] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, *Numerical Recipes in C*, 2nd edition, Cambridge University Press, 1997.